CSE 546 — Project Report

Kasturi Chandraprakash Adep
Sahil Santosh Patil
Prabhudev Terakanambi Rajendra

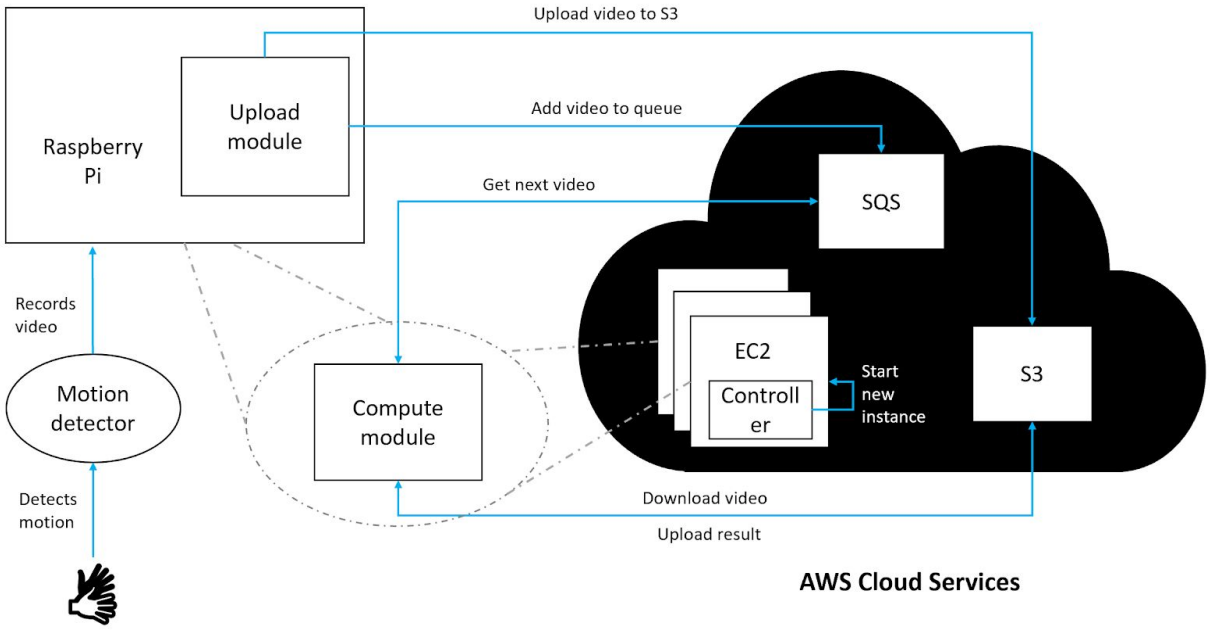
1. Problem statement

The aim of this project is to build an elastic and responsive real-time objection detection application using AWS resources and Raspberry Pi based IoT devices. For every motion detected by the sensor, videos will be recorded and darknet will be used to perform object detection on them. The results of the application will be stored on Amazon S3 along with the video input.

2. Design and implementation

2.1 Architecture

The below diagram presents an overview of the application architecture. And the following sections explain the details for each component.



Application architecture

- Raspberry Pi and motion detector – Whenever the motion detector detects a motion it triggers the camera and starts recording a video for a duration of 5 seconds. After a video is recorded it is saved in Pi and the upload module is invoked.

- EC2 – We use Amazon EC2 to perform object detection on each captured video. The design consumes a maximum of 20 t2.micro instances. Once an instance is started it invokes the compute module.

- S3 - As mentioned in the problem statement, we use Amazon S3 for storing the videos and processed results. We have decided to maintain two different buckets, a video bucket that stores all the videos. The upload module uses this bucket. The second bucket is the result bucket and is used to store a list of objects detected in the video.

- SQS - We use Amazon SQS to queue all the input video names. The compute module uses this queue to get videos from S3 for processing. The application specifically uses a FIFO queue. The intuition behind this choice is explained in later sections.

- Upload module – The upload module gets the recently recorded video and uploads it to the video bucket in Amazon S3. The video name is stored as the key and its content as the value. Simultaneously the module adds the key to a FIFO Amazon SQS.

- EC2 Controller – This component controls the number of EC2 instances running at any given point. Based on the incoming load (size of SQS), it checks for existing idle instances and starts up new instances as required. The controller runs on one of the EC2 instances. Once videos start recording, the controller polls SQS every 5 seconds. If at any point the length of the queue becomes equal to or greater than 5, the controller starts up 5 (or less depending on availability) new EC2 instances to start processing the videos.

- Compute module - This module runs independent of the other modules and executes as soon as an EC2 instance is started. It gets the next video name from the SQS and downloads it from the video bucket in S3. Next it invokes darknet to process the video. Post execution the output is parsed and the result is stored onto the result bucket in S3. On Pi, compute is invoked before recording begins. The program keeps checking the queue until it finds a video in SQS. It then picks up a video and starts processing videos iteratively until the length of the queue becomes zero. On EC2, if the length of the queue is zero for a while (35 seconds) the compute program shuts down the EC2.

2.2 Autoscaling

We ensure auto-scaling in our application using the following components. Based on the incoming load i.e. number of videos in queue the number of EC2s scales out and scales in automatically.

1. FIFO SQS - SQS is at the heart of our auto-scaling algorithm. We choose the FIFO queue against the standard implementation for the basic differences between the two. As per our design, both Pi and EC2 process videos from the SQS which may cause a race condition between the two. And so, to avoid duplicate processing of a video we use a FIFO SQS. The EC2 controller uses the queue length to determine the current load and performs the necessary amount of scaling. The compute module uses the same metric to initiate automatic scaling down.

2. Controller - The controller polls the SQS every 5 seconds. The intuition behind this choice is that it takes about 5-7 seconds for a video to record, be uploaded to S3 and added into the queue. With this approach we minimize the number of polls to the queue which is an expensive operation. Due to hardware constraints one EC2 takes about 3 minutes to process a video in addition to 30 seconds required for its initialization. Whereas Pi executes the same video in 45 seconds. To optimally use all resources available, we do not fire instances unless the current load exceeds Pi's capacity.

   We conservatively estimated that by the time EC2 completes processing one video, Pi will process about 4 videos. So if the queue has more than 4 videos waiting, we start 5 new instances simultaneously to handle them. The instances take upto 30 seconds to start and execute the compute module. The controller then waits for the instances to start and post their initialization polls the queue again. If the queue has 5 or more videos in the queue again, 5 more instances are started. Thus the controller helps scale the application based on the incoming load.

   While the 30 second wait for EC2 to start may seem like an overhead, this helps optimal use of all resources available to us.

3. Compute Module - On initializing, every EC2 instance executes the compute module. Videos are executed one by one until the length of the queue becomes zero. Once the length becomes zero, the instance waits for 35 seconds in case a new video is populated in the queue. If no new input is received the instance shuts down automatically. We wait 35 seconds to compensate for the time required to start an instance in case load is received. Thus the application automatically scales down based on the length of the SQS.

3. Testing and evaluation

Explain in detail how you tested and evaluated your application. Discuss in detail the testing and evaluation results.

We conducted the following tests to design the architecture presented in this report.

1. Time taken by Raspberry Pi to record and upload a single video to S3

   Result: 5-7 seconds

2. Time required to perform object detection on recorded video by invoking darknet on Raspberry Pi

   Result: 40-45 seconds

3. Time required to perform object detection on recorded video by invoking darknet on EC2

   Result: 3-4 minutes

4. Time required to perform object detection on 2 videos by invoking darknet on Pi using multiple threads

   Result: 60-70 seconds

5. Time required to perform object detection on 2 videos by invoking darknet on EC2 using multiple threads

   Result: 7-8 minutes

6. Time required to download video from S3 on Pi

   Result: 1-2 seconds

7. Time required to download video from S3 on EC2

   Result: 1-2 seconds

8. Time taken by EC2 to upload result to S3

   Result: 1-2 seconds

Based on the results, we concluded that by the time EC2 processes a single video, Pi will process about 4 videos. Time required to upload and download videos to and from S3 did not contribute significantly to the execution time.

We tested our application for 10 videos with 10 EC2 instances. 5 videos were processed by 5 EC2 instances (one each), whereas 5 videos were processed by Pi. The total time required was about 5 mins.

The recording for 10 videos took 70 seconds. Post which Pi executed its share of input in 3.5 minutes. The controller initialized 5 instances after 6 videos were recorded. The EC2 instances started running after approximately 30 seconds and finished uploading the results after 3.5 minutes. As the queue was empty, no new instances were started and the initial 5 stopped after waiting 30 seconds for new input.

4. Code

Explain in detail the functionality of every program included in the submission zip file.

4.1 Source Files and Functionalities

1. **detectNRecord.py :** python script to trigger the camera module to record a 5 sec video on detection of motion. It then triggers an upload of the recorded video to S3 using the uploader.sh script. It also starts a compute thread using comput.sh on raspberry pi.
2. **uploader.sh :** sets AWS credentials as environment variables and runs uploader jar file.
3. **darknet.sh :** shell script to invoke darknet commands on the given input files.
4. **compute.sh :** sets AWS credentials as environment variables and runs the compute jar file.
5. **Uploader-x.x.x.jar**
   a. Picks up a recorded video file from the local directory on pi based on the file name passed as an argument
   b. Uploads this video file to S3 video bucket
6. **Controller-x.x.x.jar**
   a. Continuously keeps monitoring the SQS queue for estimating the workload on the system
   b. Based on the load and available EC2 instances at the moment, starts the appropriate number of new EC2 instances
   c. Waits for EC2 instances to start before continuing
7. **Compute-x.x.x.jar**
   a. Continuously keeps monitoring the SQS queue for new videos to process
   b. Picks up the video name from the queue and deletes that key from the queue

c. Downloads that particular video from S3
d. Invokes Darknet program (using darknet.sh) for that video
e. Gets the output of darknet program, parses this output as required and puts it in file
f. Uploads this file(with same name as the corresponding video) to S3 output bucket
g. Starts over again

8. **user-data.txt** : Runs on EC2 boot up. It is responsible for running the compute jar on each newly started EC2 instance

4.2 Steps for installation and Execution

Installation:

1. Once you download and extract the project zip file, build the Uploader, Controller and Compute modules separately using maven build(Please use JDK 1.8). This will generate the necessary jars needed for the project. The jars will be generated in the corresponding target directory for each project module. We will need these jars in the next steps.
2. Setting up the Raspberry Pi:
    a. Set up raspberry pi camera and motion detector
    b. Copy all files from 'Scripts for Raspberry Pi' directory within the project directory to '/home/pi/demo' directory on your raspberry pi.
    c. Also, place the Uploader-x.x.x.jar and Compute-x.x.x.jar (from the previous steps) in the same directory ('/home/pi/demo')
    d. Make sure darknet and java(1.8) are installed properly.
3. Setting up EC2 (worker) instances:
    a. Create 20 EC2 instances with darknet and java (1.8) installed on it.
    b. Assign appropriate IAM role
    c. Place the compute-x.x.x.jar (from step 1) in the '/home/ubuntu/demo' directory.
    d. Copy the darknet.sh file from 'Scripts for EC2' directory within the project to '/home/ubuntu/darknet' directory.
    e. Use the content of the user-data.txt file from the 'Scripts for EC2' directory to set up user data for each EC2 instance. This will make sure that each EC2 executes the commands present in the user-data everytime it boots up.
4. Setting up EC2 controller instance:
    a. Create an EC2 instance with java (1.8) installed on it.
    b. Assign appropriate IAM role
    c. Place the controller-x.x.x.jar (from step 1) in the '/home/ubuntu/demo' directory.
5. Setting up S3 and SQS:
    a. Create two S3 buckets, one for videos and other for output
    b. Create SQS FIFO queue

Note:

- After placing all the scripts and jars in the appropriate locations, update the permissions of every file with execute permission.

    sudo chmod +x <filename>

Steps to Run:

1. Boot up the raspberry pi and EC2 controller instance.
2. Run the controller jar present in the EC2 controller instance.

   sudo java -jar controller-x.x.x.jar

3. Run the detectNRecord.py

   sudo python detectNRecord.py